One Billion Apples' Secret Sauce : Recipe for the Apple Wireless Direct Link Ad hoc Protocol

Stute, Milan; Kreitschmann, David; Hollick, Matthias

(2018)

One Billion Apples' Secret Sauce: Recipe for the *Apple Wireless Direct Link* Ad hoc Protocol

Milan Stute
Secure Mobile Networking Lab
TU Darmstadt, Germany
mstute@seemoo.de

David Kreitschmann
Secure Mobile Networking Lab
TU Darmstadt, Germany
dkreitschmann@seemoo.de

Matthias Hollick
Secure Mobile Networking Lab
TU Darmstadt, Germany
mhollick@seemoo.de

ABSTRACT

Apple Wireless Direct Link (AWDL) is a proprietary and undocumented IEEE 802.11-based ad hoc protocol. Apple first introduced AWDL around 2014 and has since integrated it into its entire product line, including iPhone and Mac. While we have found that AWDL drives popular applications such as AirPlay and AirDrop on more than one billion end-user devices, neither the protocol itself nor potential security and Wi-Fi coexistence issues have been studied. In this paper, we present the operation of the protocol as the result of binary and runtime analysis. In short, each AWDL node announces a sequence of Availability Windows (AWs) indicating its readiness to communicate with other AWDL nodes. An elected master node synchronizes these sequences. Outside the AWs, nodes can tune their Wi-Fi radio to a different channel to communicate with an access point, or could turn it off to save energy. Based on our analysis, we conduct experiments to study the master election process, synchronization accuracy, channel hopping dynamics, and achievable throughput. We conduct a preliminary security assessment and publish an open source Wireshark dissector for AWDL to nourish future work.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Ad hoc networks**; **Link-layer protocols**;

KEYWORDS

AWDL, Reverse engineering, Ad hoc networks, IEEE 802.11, Proprietary protocol, Apple, macOS, iOS

MobiCom '18, October 29–November 2, 2018, New Delhi, India

ACM Reference Format:
Milan Stute, David Kreitschmann, and Matthias Hollick. 2018. One Billion Apples' Secret Sauce: Recipe for the *Apple Wireless Direct Link* Ad hoc Protocol. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29–November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3241539.3241566

1 INTRODUCTION

Apple Wireless Direct Link (AWDL) is a proprietary protocol deployed in about 1.2 billion[1] end-user devices consisting of Apple's main product families such as Mac, iPhone, iPad, Apple Watch, and Apple TV—effectively all recent Apple devices containing a Wi-Fi chip. Apple does not advertise the protocol but only vaguely refers to it as a "peer-to-peer Wi-Fi" technology [5, 6]. Yet, it empowers popular applications such as AirDrop and AirPlay that transparently use AWDL without the user noticing. We believe that public knowledge of this undocumented protocol would be beneficial for the following reasons: First, since AWDL is based on IEEE 802.11, there are potential performance and co-existence issues that need to be identified. This is especially important in regulated environments as AWDL uses various channels and employs a channel hopping mechanism that might interfere with corporate Wi-Fi deployments. Second, the Wi-Fi driver (where AWDL is implemented) is the largest binary kernel extension in current versions of macOS. Given the recently published vulnerabilities in Wi-Fi chip firmware [7, 8] that might lead to full system compromise [9], we highly recommend a security audit of the protocol and its implementations as vulnerabilities in non-standardized protocols are even more likely to occur. For example, protocol fuzzing requires knowledge of the frame format. Third, an open re-implementation of the protocol would allow interoperability with other operating systems, eventually enabling high-throughput cross-platform direct communication. Such technology is required, for example, in smartphone-based emergency communication applications [21, 25].

To maximize the impact for the research community, we have lifted a layer in Apple's ecosystem and unveiled an

[1] Based on unit sales for iPhone, iPad, and Mac since 2014 [4].

existing yet obscure wireless ad hoc protocol. In this paper, we conduct a comprehensive investigation on AWDL by means of binary and runtime analysis, and present its frame format and operation. In short, AWDL is based on the IEEE 802.11 standard and makes use of vendor-specific extensions that allow custom protocol implementations. Each AWDL node periodically emits custom action frames containing a sequence of Availability Windows (AWs) indicating its readiness to communicate with other AWDL nodes. An elected master node synchronizes these sequences. Within these AWs, nodes are able to communicate with their neighbors using a dedicated data frame format. Outside the AWs, nodes can tune their Wi-Fi radio to a different channel to communicate with an access point, or turn it off to save energy. We summarize our main contributions:

- We provide insights into the macOS operating system and its Wi-Fi driver architecture and debugging facilities to help future research endeavors (Section 3).
- We present the AWDL frame format and operation in detail (Sections 4 to 6).
- We conduct an experimental analysis of AWDL to assess election behavior, synchronization accuracy, throughput, and channel hopping strategies (Section 7).
- We discuss protocol complexity, energy efficiency, and perform a preliminary security assessment where we report a security permission problem in a macOS kernel extension (Section 8).
- We publish an open source AWDL Wireshark dissector [23].

Furthermore, we give background on related direct wireless communication technologies in Section 2 and conclude this work in Section 9.

2 BACKGROUND

AWDL has been referenced in several patents such as [24] and can be classified as a wireless ad hoc protocol which allows peers to communicate directly with each other. There exist already a number of other technologies which we summarize in the following.

IEEE 802.11 IBSS. The IBSS mode commonly known as "ad hoc" mode creates a distributed wireless network without special controller roles. An IBSS is created by sending beacon frames with an SSID and BSSID on a particular channel. Other nodes joining the network will send out beacons themselves using the same information. The mode is robust to nodes leaving the network as all nodes broadcast beacons. The nodes do not require any further synchronization. However, IBSS has never become widely deployed, mostly due to lack of efficient power saving mechanisms, which are crucial for mobile devices [11]. Flawed implementations are another common problem [29]. IBSS is not supported on Android

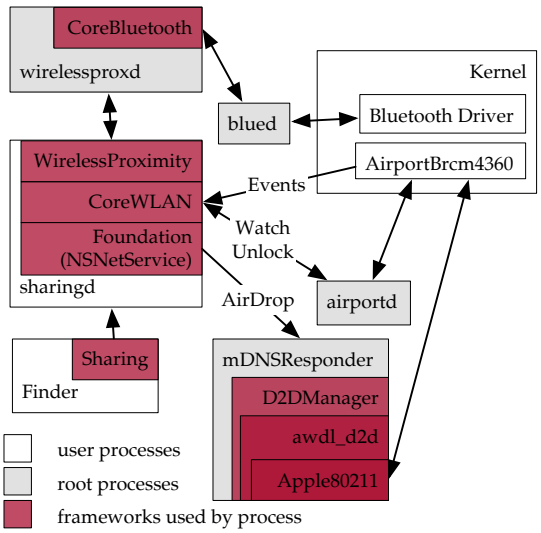

Figure 1: Interaction of different macOS processes and frameworks used for controlling AWDL.

and Microsoft announced it might not be available in future versions of Windows [27]. On Apple's operating systems encryption is not supported and iOS only allows to join existing IBSS networks.

Wi-Fi Peer-to-Peer. Wi-Fi P2P [36], also known under its certification name Wi-Fi Direct[2], allows connecting multiple devices directly without a base station. During operation, one node assumes the role of a Group Owner (GO) which closely resembles infrastructure (or BSS) operation. It is not possible to migrate the role of the GO to another device: if the GO leaves the network, a new network must be created. Wi-Fi P2P connections are established by listening on one channel and sending probe requests on all channels. This delays the connection process in practice. Experiments show that establishing a connection takes from four to more than ten seconds [11]. Discovering devices thus drains their battery very fast.

Tunneled Direct Link Setup. Tunneled Direct Link Setup (TDLS) is an IEEE 802.11 extension that enables direct communication between two nodes in the same BSS. In networks without TDLS, all traffic passes the Access Point (AP) even when the two communicating nodes are within communication range. TDLS requires both nodes to be connected to the

[2]The Wi-Fi Alliance is a vendor association which holds the Wi-Fi trademark for IEEE 802.11-based technologies and certifies products using the specification. Although the Wi-Fi Alliance does not formally create the standard, their certification has relevance in the market. The alliance also creates their own standards based on IEEE 802.11 such as P2P and NAN.

same AP since control frames are tunneled through the AP and, thus, cannot be used in real ad hoc scenarios.

Neighbor Awareness Networking. Neighbor Aware Networking (NAN) [35], also known as Wi-Fi Aware, extends IEEE 802.11 with proximity service discovery. NAN is designed to be energy efficient, allowing continuous operation on battery-powered devices [12]. NAN is supported in Android 8 [17], but we did not find any devices with compatible hardware. NAN depends on beacon frames sent from an elected master. These synchronize the timing of all devices in an area. During a short discovery window the master sets, devices can turn their radio on, exchange service and connection information (e.g., parameters for Wi-Fi P2P) and turn their radio off again. In fact, we found that AWDL employs similar concepts as NAN, but the actual implementation differs strongly from that of NAN. In addition, NAN does not feature a data path for transmission of user data.

Bluetooth. Bluetooth [10] is a separate standard with different PHY and MAC layers. Bluetooth operates in the 2.4 GHz band as Wi-Fi and is often integrated into the Wi-Fi chip to share the same antennas. Bluetooth Low Energy (BLE) is incompatible with classic Bluetooth and is optimized for low energy consumption and, therefore, offers limited bandwidth. The usable maximum BLE 4.2 data rate is 394 kbit/s [14]. It is commonly implemented in small battery-powered devices such as smartwatches and fitness trackers. BLE is not designed for large data transfers but can be used for bootstrapping high-bandwidth links such as AWDL.

3 METHODOLOGY

Reverse engineering is more of an art than a science and, hence, it is hard to write generic recipes. Nevertheless, we structure our methodology for reversing closed-source network protocols with a focus on the macOS operating system so that it can be used in related research endeavors. In the following, we describe how binary and dynamic runtime analysis in tandem can result in full disclosure of the workings of a complex wireless network protocol. Previous exemplary works have reverse engineered the Skype protocol [26], Broadcom Wi-Fi chip firmware [28], and the Fitbit ecosystem [13].

3.1 Binary Analysis

We analyzed numerous binaries related to AWDL to finally find those parts that implement the protocol. We first illustrate our selection process and then discuss the two-part Wi-Fi driver which implements most of the AWDL protocol stack. We focus our analysis on macOS and assume that the architecture is in principle similar to that of iOS. We used a decompiler to analyze the target binaries.

Binary Selection. Apple excessively uses *frameworks* and *daemons* in its OSes. Consequently, there are numerous dependencies which result in a complex binary selection process. Frameworks offer an API to their corresponding singleton daemons and can be used by other daemons and processes. We started off by crawling the system for binaries that had "802.11", "Multicast DNS (mDNS)," or "sharing" in their names. We found more related targets by following dependencies. We show part of the discovered dependencies and interactions in Fig. 1. While there are user-facing binaries such as the `sharingd` daemon, the most relevant binaries reside in the kernel, in particular, the generic Wi-Fi driver `IO80211Family` and the device-specific variants `AirportBrcm4360` and `AirportBrcmNIC`. Each of them includes hundreds of AWDL-related functions, suggesting that the bulk of the protocol stack is implemented here. We found that `IO80211Family` takes care of most of the AWDL frame parsing and creation as well as maintaining the AWDL state machine. The device-specific driver handles time-critical functions such as synchronization. As both driver parts are among the largest kernel extensions present in macOS , understanding internal driver structures were key to make sense of the decompiled code.

Finding Interesting Code Segments. Due to the size of the macOS Wi-Fi driver, we needed to quickly find functions that would implement part of the AWDL protocol. Fortunately, Apple does not strip symbol names from their binaries, such that searching for "awdl" in the symbol table (e. g., using `nm`) results in a number of hits. Some of those symbols additionally contain "parse" and "TLV" in their name (e. g. `parseAwdlSyncTreeTLV`) which helped us understand the calculation of some Type-Length-Value (TLV) fields. Furthermore, debug log statements give hints about the purpose of a code segment inside a function. Therefore, we can search for debugging strings and their cross-references to find details such as the misalignment threshold in Section 6.2.

Leaked Broadcom Driver Source Code. As another source of information, we used a dated Broadcom Wi-Fi driver whose source code was leaked [15]. We found several references to AWDL in the source code but none of the core functionality. We suspect that Broadcom uses a modular firmware concept with one central repository for a wide range of features. Special features such as AWDL are made available selectively to their customers such as Apple. More important than the references to AWDL are some C structs found in the source code. These include key structures such as the Synchronization Parameters TLV and Channel Sequence TLV (more in Section 5). The leaked code also contains the source code for the `wl` utility, which provides debugging features for the driver and is further discussed in Section 3.2.

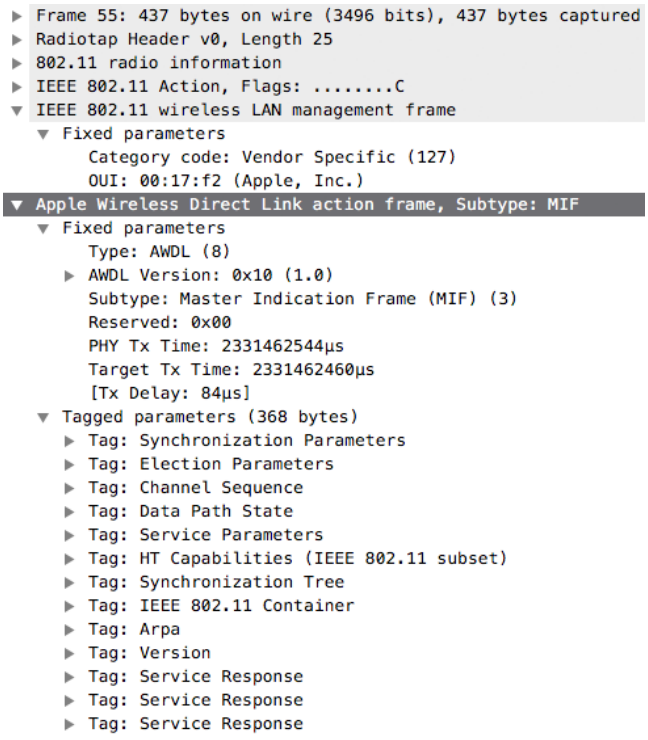

Figure 2: Screenshot of our Wireshark dissector.

Dissecting Structures. To understand the driver's functions, we needed to reconstruct the underlying data structures. The leaked source code shows that most of the AWDL-related functions use an `awdl_info` struct as a first parameter. The `wlc_dump_awdl` function prints internal data in a readable format and, thus, was an ideal target to reconstruct the internal structures as shown below:

```
bcm_bprintf(a2, "AWDL master home channel = %d\n",
            awdl_info->master_home_channel);
```

The result of our binary analysis was a complete Wireshark dissector for AWDL that we also used for the dynamic analysis of the protocol and for evaluating our experiments. We show our dissector in Fig. 2.

3.2 Runtime Analysis

The complete protocol operation was difficult to comprehend with the binary analysis alone. To understand the semantics of synchronization, election, service discovery, and data path, we complemented our static analysis with a dynamic approach. In this section, we discuss dedicated macOS logging and debugging facilities that helped to analyze the protocol. In particular, we used the *Console* application, the `ioctl` interface, the leaked Broadcom `wl` utility, as well as Apple's undocumented *CoreCapture* framework. The latter is especially verbose but required us to write an additional dissector for Wireshark as it uses a private data format.

Apple Console. The *Console* program is the central place to access logs since macOS 10.12 and includes debug messages from the kernel. To receive verbose output from the Wi-Fi driver, we increased the log level using custom boot arguments which we found by searching for references to the `PE_parse_boot_arg` function in the Wi-Fi driver. The following boot arguments maximize the driver's debug output:

```
nvram boot-args="debug=0x10000 \
      awdl_log_flags=0xffffffffffffffff \
      awdl_log_flags_verbose=0xffffffffffffffff \
      awdl_log_flags_config=1 wlan.debug.enable=0xff"
```

With the increased log level, *Console* shows additional information such as state transitions and the current channel sequence:

```
IO80211Family <...> com.apple.p2p: AWDL ON: [infra
    ↪ (100) 72%], (6/44/44) [44 0 0 0 0 0 0 0 6 44
    ↪ 44 0 0 0 0 0] Low Power
```

`ioctl` Interface. `ioctl` system calls are a standard way to communicate with devices on Unix-based systems. Apple uses `ioctl`s to configure wireless interfaces such as associating with an AP or creating an IBSS. Apple provides the header files with the request format, the available request types, and the data structures for macOS 10.5. These old header files can be brought up to date using information from the binary analysis. The `apple80211VirtualRequest` method contains calls to all handler functions. Out of the available 72 request IDs, 40 relate to AWDL. These requests can set several parameters in the driver. Especially useful is the card-specific `ioctl`. It allows wrapping a Broadcom-specific `ioctl` inside an Apple `ioctl`, providing us with a direct interface with the Broadcom driver. Note that it is no longer possible to send Broadcom-specific `ioctl`s since Apple fixed our reported vulnerability (Section 8): the driver now checks for a private *entitlement* security permissions [2] (`com.apple.driver.AirPort.Broadcom.ioctl-access`) which requires a binary signed with an Apple private key. It should be possible to overwrite the respective permission-checking function in the driver using a kernel extension patching framework such as [16] to restore unrestricted `ioctl` access. Driver patching requires disabling Apple's *System Integrity Protection* [1].

Broadcom `wl` Utility. The Broadcom `wl` utility found in the leaked source code provides several methods to access internal information about AWDL operations, which are directly related to the structures found during binary analysis. Although the AWDL-specific driver code was missing in the leaked source code, the `wl` source code contains AWDL related commands and structures. `wl` allows us to query the

current AWDL driver status using commands such as `dump awdl` and `awdl_advertisers`. The latter shows information about neighboring nodes including RSSI.

CoreCapture Framework. CoreCapture is Apple's primary logging and tracing framework for IEEE 802.11 on iOS and macOS. CoreCapture combines raw protocol traces with traditional log entries and provides snapshots of the device and driver state. CoreCapture is undocumented but was referenced in a `dumpPacket` function that we found in the driver. Since the framework outputs (among other logs and memory dumps) numerous PCAP trace files with a custom header format, we wrote a Wireshark dissector for CoreCapture that we make available to the public [23]. In addition, we publish a manual for CoreCapture with this paper [22].

4 AWDL OVERVIEW

Based on our analysis, we formulate *hypotheses* regarding the design goals and decisions of AWDL: (*i*) leverage existing hardware (Wi-Fi chip), thus building the protocol on top of IEEE 802.11; (*ii*) conserve energy, especially on mobile devices, hence synchronizing and putting the Wi-Fi chip into a power-saving mode during idle times; (*iii*) allow seamless operation of direct and infrastructure-based communication, so enable synchronized channel hopping without disconnecting from an AP; and (*iv*) enable fast service discovery, thus offloading DNS-SD to Wi-Fi frames. Commodity Wi-Fi chips usually have a single RF chain and are, therefore, restricted to a single wireless channel at any given time. To use multiple channels, an adapter needs to switch channels and cannot use the regular wireless connection for short periods of time. This is expected behavior for roaming (scan for available networks while being connected to a network) and power saving features (switch off the radio). To use these short periods for data transfer, devices need a method for discovery and coordination when to meet on which channel. We depict the main AWDL phases in Fig. 3 and briefly introduce them in the following.

(1) Activation. Apple uses AWDL as an on-demand communication technology. This means that AWDL is inactive by default, but applications can (temporarily) request activation. For example, AirDrop uses BLE for activation by sending truncated hashes of the user's contact information; AirPlay receivers (Apple TV) constantly announce their presence via AWDL; and third-party application may activate the interface indirectly by advertising services via the `NSNetService` API [6].

(2) Master Election. Apple uses fixed social channels (6, 44, and 149[3]) for coordination using Periodic Synchronization Frames (PSFs). A node starting its AWDL interface monitors

[3]depending on the country

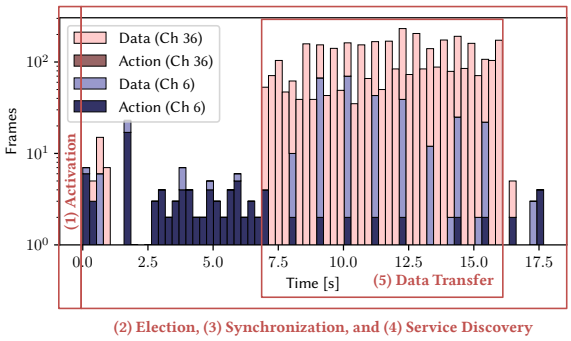

Figure 3: The five main AWDL phases. Exemplary trace showing a 100 MiB file transfer via AirDrop including activity on multiple channels and differentiating the traffic types.

the social channels for some time to discover other nodes in range. If AWDL Action Frames (AFs) are received, the node can adopt an existing master. If no frames are received, it assumes the master role itself. We elaborate on the election process in Section 6.1.

(3) Synchronized Channel Sequences. AWDL is built around a sequence of time slots (Availability Windows (AWs) and Extended Availability Windows (EAWs)). For each of these slots, peers broadcast if they are available for AWDL data and, if so, on which channel they will be. Peers match these advertisements with their own AW sequence. If there is a common channel in a particular AW, communication during this AW is possible. A synchronization mechanism aligns the sequences between nodes. We elaborate on the synchronization and channel alignment processes in Sections 6.2 and 6.3, respectively.

(4) Service Discovery. DNS Service Discovery (DNS-SD) [18] also known as "Bonjour" can be offloaded to AWDL. AWDL piggy-backs DNS-SD responses directly onto its AFs such that services are immediately discovered whenever a node changes its advertisements. For space reasons, we do not elaborate on the service discovery component in this paper.

(5) Data Transfer. AWDL uses a vendor-specific frame format header for user data which exclusively transports IPv6 packets. When transmitting user data to a particular peer, a node needs to calculate the AWs during which both nodes are tuned to the same channel and only transmit frames during those AWs. In addition, AWDL adapts its channel sequence according to the current outgoing traffic load. We discuss the data transfer mechanisms in detail during the experimental evaluation in Section 7.

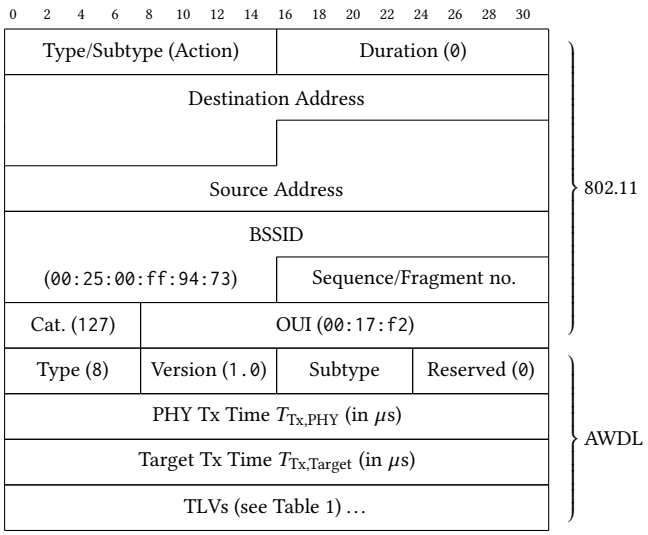

Figure 4: AWDL action frame.

Table 1: TLVs used in AWDL. We give the name and type value of a TLV, indicate whether it is included in PSFs or MIFs (✓), and whether it can be present multiple times (+).

Name	Type	PSF	MIF	Purpose
Sync. Parameters	4	✓	✓	Election and Synchronization
Channel Sequence	18	✓	✓	
Election Parameters	5	✓	✓	
Election Parameters v2	24	✓	✓	
Synchronization Tree	20	✓	✓	
Service Parameters	6	✓	✓	Service Discovery
Service Response	2		✓+	
Arpa (Reverse DNS)	16		✓	
Data Path State	12	✓	✓	User Data Transmission
HT Capabilities	7		✓	
VHT Capabilities	17		✓	
Version	21	✓	✓	Compatibility

5 FRAME FORMAT

We discovered two general frame types used by AWDL: *action* and *data* frames which are used for coordination and direct data transfer, respectively. We elaborate on the frame format of these types in the following.

5.1 Action Frames

AWDL uses IEEE 802.11 vendor-specific AFs which generally allow vendors with an Organizational Unique Identifier (OUI) to implement IEEE 802.11 frames with arbitrary payloads [31]. The AWDL vendor-specific extension consists of a fixed-sized header and multiple TLV fields as shown in Fig. 4. A TLV consist of a 1-byte *type* field, followed by a 2-byte *length* field which indicates the length of the subsequent *value* byte string. The fixed header mostly includes static values such as AWDL-specific BSSID, OUI, version, and type. The two timestamps indicate when the frame was created and, therefore, at which time the included information was up-to-date ($T_{Tx,Target}$), and when it was actually queued for transmission ($T_{Tx,PHY}$). Their difference approximates the sender's transmission delay and is used for synchronization purposes. There are two AWDL AF subtypes: Periodic Synchronization Frame (PSF) and Master Indication Frame (MIF). These frame types start with the same fixed header and differ only in the included set of TLVs and, hence, their size. We show the frame format excluding the FCS at the end of the frame in Fig. 4. We first explain the purpose of the subtypes and then discuss TLVs used in AWDL.

Periodic Synchronization Frame (PSF). The PSF is used for synchronization and is further explained in Chapter 6.2. The name was gathered from a patent [33]. Its subtype is 0. If all participating devices support the 5 GHz band, the PSF is the only frame type also seen on the 2.4 GHz band.

Broadcast Master Indication Frame (MIF). The MIF is used for multiple purposes, e.g., election (Section 6.1) and service discovery. It includes more TLVs and is sent by all devices in the network regularly. The MIF subtype is 3.

TLVs. TLVs contain the actual control information. The different types can be attributed to one of the following purposes: *election and synchronization*, *service discovery*, and *user data transmission*. In addition, the *version* TLV provides a 1-byte version number which presumably supersedes the version field in the fixed header (see Fig. 4). We summarize all TLVs in Table 1 and discuss them briefly in the following. The names were taken from function names and debugging strings found during binary analysis. We discuss only some TLVs in detail in this paper, and refer to our Wireshark dissector for the full specification. Note that some *type* values (e. g. 1, 3, and 8) are missing in Table 1. These types appear to be deprecated as they were not actively used in the AWDL versions that we analyzed.

The *election and synchronization* processes handle the overall cooperation of the devices. The data in these TLVs determines, e. g., which node takes the master role and which channels are to be used. Curiously, the Synchronization Parameters TLV includes its own channel sequence, so the separate Channel Sequence TLV appears to be redundant. It was however always transmitted on current operating system versions. This is further discussed in Sections 6.1 to 6.3. The *service discovery* components offload mDNS and DNS-SD functionality to the AFs. They contain the hostname (Arpa TLV); and PTR, SRV, and TXT resource records (Service Response TLV). The *user data transmission* components

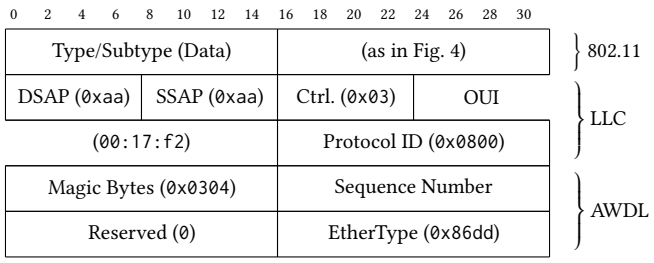

Figure 5: AWDL data frame header.

are used to negotiate the parameters for direct connections between devices. For example, supported PHY rates are announced in the HT/VHT Capabilities TLVs which are similar to the ones introduced in the IEEE 802.11 n and 11 ac amendments [31]. In addition, each peer announces in the Data Path State TLV the Wi-Fi network (BSSID) that it is currently connected to as well as the real MAC address of the Wi-Fi chip. We believe that this information could be used to offload an AWDL connection to an infrastructure network if both peers are connected to the same network. However, this would require additional reachability tests due to network policies such as client isolation, and we did not observe such behavior in practice. The *version* TLV includes the AWDL version (half a byte for major and minor version number each) as well as a device class ID. We found that v3.x is used in macOS 10.13 and iOS 11; and v2.x in macOS 10.12 and iOS 10 (and potentially prior iOS versions). AWDL v1.x is used in macOS 10.11 which does not support the *version* TLV. The device class seems to indicate the OS type of the node, e. g., macOS (1) or iOS (2).

5.2 Data Frames

AWDL uses IEEE 802.11 data frames for user data transmission. The To-DS and From-DS flags are set to zero, similar to IBSS which means that these frames are addressed directly, and three address fields are used for the destination, source, and BSSID. We depict the AWDL data frame format in Fig. 5. The BSSID in AWDL frames is always `00:25:00:ff:94:73` which belongs to the OUI `00:25:00` that is assigned to Apple [20]. The LLC header contains a different Apple OUI (`00:17:f2`) and a protocol ID in the SNAP part. These headers are part of the IEEE 802 standard [30] and allow vendors to implement their own protocols on higher layers. The actual AWDL data header essentially consists of a sequence number and the EtherType of the transported protocol. We identified IPv6 as the only protocol used with AWDL.

5.3 Addressing for Higher-Layer Protocols

AWDL is used in conjunction with higher-layer protocols. Therefore, it needs some way to address AWDL nodes via a network layer protocol. This is especially important because AWDL implements privacy-enhancing MAC randomization which means that instead of using the Wi-Fi chip's fixed MAC address, it generates a random address every time the interface is activated. In IPv6, address resolution is usually done via the Neighbor Discovery Protocol (NDP). Apple, however, does not use NDP for AWDL, but instead generates link-local IPv6 addresses from the source address field contained in the AFs (Fig. 4) using a method described in RFC 4291 [19, Appendix A]. This method constructs a link-local IPv6 address based on the 48-bit MAC address of the network interface. In particular, given a 48-bit MAC address $o_0:o_1:o_2:o_3:o_4:o_5$, the corresponding link-local IPv6 address is constructed as:

`fe:80::`o_0`^0x02:`o_1`:`o_2`:ff:fe:`o_3`:`o_4`:`o_5.

Using this standardized method, nodes can add their neighbors to the neighbor table immediately after receiving the first AF without the need or overhead of an additional address resolution protocol such as NDP or ARP.

6 PROTOCOL OPERATION

We present the detailed mechanisms that are used to form and maintain an AWDL *cluster*. In particular, we discuss how a master is elected, and conflicts are resolved; how nodes synchronize their clock to the master; and, finally, how the announced channel sequence maps to the sequence of AWs.

6.1 Master Election

In this section, we explain the election process and the tree-based synchronization structure. In particular, we focus on the mechanisms that make AWDL robust to master nodes leaving or joining the cluster.

Role of the Master Node. As already mentioned, AWDL relies on roughly synchronous clocks of all participating nodes in a cluster. To achieve this, it is paramount that there is exactly one node in the cluster which has the responsibility of emitting a "clock signal." This is the one (and as far as we know the only) role of the *master* node. All other nodes in the cluster are called *slaves* and should adopt this signal. In a simple scenario with only two nodes, one node will be the master and another a slave. In larger scenarios, slave nodes might be more than one hop away from the master node. In such cases, intermediate slave nodes will take the role of *non-election masters*, which have the responsibility to repeat the master's clock signal. The intermediate master nodes are included in the Synchronization Tree TLV where each node announces the path to the "top" master. In any case, there is only one top master in a cluster.

Master Metric. The master election is based on a *metric* field which is included in the Election Parameters v2 TLV. The node that announces the largest metric value will become the

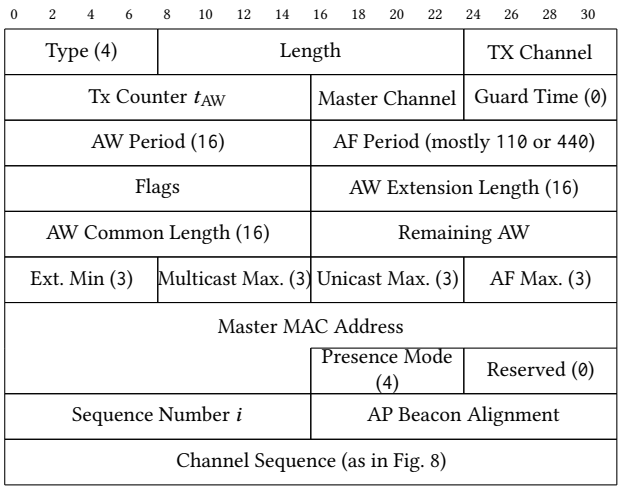

Figure 6: Synchronization Parameters TLV

master of that cluster. Apple's patent [34] claims that these metrics could be based on available energy resources, CPU load, signal strength, etc. In practice, however, the metric is simply chosen at random. A node that activates its AWDL interface initially sets its metric field to 60 and listens on the social channels for an existing master for 2 seconds. If no master is found, it draws a random number from a predefined range and sets this as its metric. We have found that this range depends on the AWDL version, e. g., 405 to 436 in v2.x and 505 to 536 in v3.x. We assume that this is done for backwards compatibility so that the master node is guaranteed to be a node running the most up-to-date version in a cluster and future protocol extensions can be supported.

Merging Clusters with Different Masters. When two already established AWDL clusters with different master nodes move into proximity, they need to merge such that nodes in the different clusters will be able to discover each other. In AWDL, the process is straight-forward as all nodes advertise their current master metric in the Election Parameters TLV. If two nodes with different masters discover each other, they receive the top master metric of the other cluster and can immediately adopt the master with the higher metric. The remaining nodes in the "lower" cluster then follow as soon as the first node advertises the new master metric.

Loop Prevention. When creating such an election tree hierarchy with multiple levels of sub-masters, loops may occur. To prevent loops and limit the maximum depth of the election tree, each AF contains a list of all nodes up to the top master in the Synchronization Tree TLV. Each node can then make sure that it does not adopt a non-election master if it is already in that node's path.

Re-Election. The initialization of a device using a low metric will prevent most random re-elections when new devices join the network. As there is no sign-off message, a master leaving the network simply stops sending AFs. Therefore, a missing master can only be detected by other devices after a certain *no master* timeout which is fixed to 96 AWs (≈ 1.5 s). Another node will then take the place of the old master. As this node was already in sync with the old master, other slave nodes do not need to re-synchronize but simply adopt the new master. In other words, AWDL is robust to "master churn," i. e., a leaving master does not interrupt communication, and a new master is seamlessly adopted. This is in contrast to other technologies such as Wi-Fi Direct, where the respective master node essentially acts as an AP which takes care of relaying data between two nodes and a leaving master would require a group re-establishment.

The Role of RSSI. The RSSI values of received AFs are used to filter out possibly unstable connections. In particular, AWDL nodes drop frames when the RSSI is below a so-called *edge sync* threshold which is set to -65 (or -78 if AirPlay is used). Frames from the current master node are accepted with a lower RSSI. These frames receive a bonus *slave sync* threshold of 5. Lowering the threshold for the master frames allows for a certain variance in the RSSI. We assume that this was done to reduce "master flapping" where a node frequently adopts a new master because it regularly drops frames and the *no master* timeout occurs.

6.2 Synchronization

Synchronization is tightly coupled with the election process since nodes always try to synchronize to their elected master. In this section, we describe how time is structured in AWDL and how nodes align their time reference with that of their master. We introduce the concept of Availability Windows (AWs), that is, short fixed-length time slots during which communication is possible. These windows have a static length, but can be extended using Extension Windows (EWs). Finally, we show how the start of an AW is determined using fields from the Synchronization Parameters TLV. We summarize the key concepts and variables in Fig. 7.

Availability Window. AWs indicate a period of time during which a device will be available for communication. These windows need to be synchronous for all nodes in a cluster such that every device starts an AW at the same time. Timing in AWDL is based on Time Units (TUs) where 1 TU = 1024 µs [31, page 141]. In the AWDL implementation, an AW is always set to be 16 TUs long. The length of an AW and all other "static" values presented in this section are contained in the Synchronization Parameters TLV. In theory, different configurations are possible, but we found that only fixed values are used.

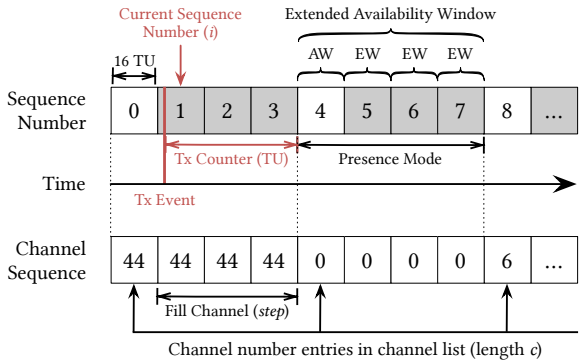

Figure 7: Structure of AWs and mapping to channel sequence.

Presence Mode and Extension Windows. For reduced power consumption, a peer can indicate that it is not listening in every AW. A presence mode p of 4, which is the only value used in Apple's AWDL implementation, means that a peer is only listening for every fourth window. If a node is transmitting or receiving data, it may extend its time spent on the channel. This is called an Extension Window (EW). A presence mode of 4 leaves space for three EWs of 16 TUs. In addition, AWDL allows to configure different numbers of unicast, multicast, and AF EWs, but these fields are currently always set to 3 and, thus, align with the presence mode. Figure 6 shows the parameters transmitted in the Synchronization Parameters TLV. Given the static configuration, the effective smallest time unit in use is four consecutive AWs/EWs. For the remainder of this paper, we use the term Extended Availability Window (EAW) to refer to such a 64 TU time slot.

Calculating the Start of an Availability Window. Each slave node needs to synchronize its clock to that of its master node. To achieve this, the master node announces the *start of the next AW*. When transmitting an AF, the master includes the number of TUs to the next EAW t_{AW} as well as the sequence number of the current AW or EW i. We mark these values in red in Fig. 7.

As these values are set when the frame is created in the driver, some time passes until the frame is actually transmitted via the Wi-Fi interface. AWDL tries to compensate for this transmitter delay by including two additional timestamps in the fixed header of each AF: the PHY and target transmission times $T_{\text{Tx,PHY}}$ and $T_{\text{Tx,Target}}$, respectively. Ideally, $T_{\text{Tx,Target}}$ is set when the frame is created, and $T_{\text{Tx,PHY}}$ just before the frame is transmitted via the interface. In the macOS driver, however, both timestamps are set in the Wi-Fi driver and, therefore, do not account for delays induced by the distributed coordination function (DCF) which controls medium access [31]. Nevertheless, a device receiving an AF

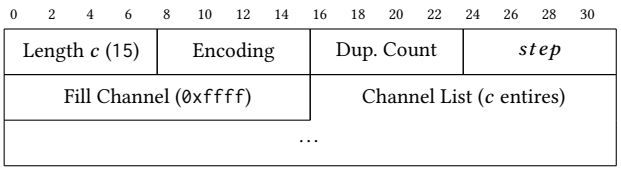

Figure 8: Channel Sequence

from its master at time T_{Rx} can approximate the start of the next AW T_{AW} as follows:

$$T_{\text{AW}} = t_{\text{AW}} \cdot 1024 - (T_{\text{Tx,PHY}} - T_{\text{Tx,Target}}) + t_{\text{air}} + T_{\text{Rx}}. \quad (1)$$

In fact, AWDL ignores the airtime t_{air} since it is in the order of sub-μs in a typical close-range Wi-Fi scenario, and the accepted synchronization error is 3 ms.[4] We experimentally evaluate the achievable accuracy in Section 7.

6.3 Channel Sequence

The AWDL channel sequence announcement builds upon the synchronized AWs and indicates whether a node is actually available for communication and, if so, on which channel it has tuned its radio. In this section, we explain how the channel sequence maps to the sequence of AWs.

The channel sequence maps channel numbers to AW sequence numbers. While the channel sequence included in the TLVs shown in Fig. 8 contains a fixed number of $c + 1 = 16$ channel entries, the sequence can be prolonged with the *step* field similar to the presence mode, so that one channel entry can span multiple AWs and EWs.[5] Setting *step* to 1 means that the channel will be active for one additional AW. However, Apple always sets this field to 3, meaning that the channel will be active for four AWs or one EAW. Thus, the channel sequence is fully aligned to the presence mode in the Synchronization Parameters TLV. Given an encoded channel sequence and an AW sequence number i, an AWDL node can calculate the currently active channel C for any peer based on the following calculation:

$$C = i \mod ((c + 1) \cdot (step + 1)) \quad (2)$$

As Apple uses fixed values for c and *step*, the announced channel sequence covers $(15+1)\cdot(3+1) = 64$ AWs which takes about one second ($64\,\text{AW} \cdot 16\,\text{TU/AW} = 1048576\,\mu\text{s} \approx 1\,\text{s}$).

7 EXPERIMENTAL ANALYSIS

We analyze the runtime behavior of AWDL in different scenarios to (*i*) validate our findings of the previous sections

[4] In the function `awdl_recv_action_frame`, a misalign metric is increased if the difference between a projection from a previous calculation and new calculation of T_{AW} is larger than 3 ms.

[5] Note that the extension with *step* works only if the *fill channel* field is set to `0xffff`, which was the case in all our captured frames.

and (*ii*) assess the performance of the protocol. First, we describe our test setup. Then, we look at the master election and synchronization accuracy in an idle scenario without data transmissions. We further analyze the channel hopping behavior and throughput performance.

7.1 Test Setup

Our test setup consists of one monitoring device and a number of Apple devices. Our monitor device is an APU board[6] equipped with two Qualcomm Atheros QCA9882 Wi-Fi cards to support simultaneous sniffing on two different channels which are tuned to AWDL's primary (44) and secondary (6) channel. Both Wi-Fi cards support hardware timestamping which mitigates variable delays in the receiver's OS stack. To synchronize the internal clocks of the sniffing Wi-Fi chips, we start each experiment with a calibration phase: we tune both chips to a common channel and let them record multiple frames. Post-experiment, we calculate the timestamp difference of frames that were received by both cards. We use the median difference to correct the clock offset and align both traces. All following experiments were conducted inside a Faraday tent to minimize interference. Our test devices include an iPhone 8 (iOS 11.2.2), an iPad Pro 10.5" (iOS 11.0.3), an iMac (Late 2012, macOS 10.12.6), and a MacBook Pro (Late 2015, macOS 10.12.6).

7.2 Master Election

In our first experiment, we analyze the master election process. We observe an AWDL cluster in an idle state, meaning that no data transmission takes place and the only observed frames are AFs. We use a setup consisting of an iPhone, iPad, iMac, and MacBook. We activate the AWDL interface by selecting the *sharing panel* in one device which causes a BLE scan and activates the AWDL interface of other devices in range (on iOS, this only works if the device is unlocked). To get more interesting results, we let the different devices join approximately 30 s after one another.

Figure 9 shows the currently selected master of each node. First, the iMac creates the AWDL cluster and consequently selects itself as the master. As soon as the iPhone joins, it takes over the master role, and the iMac adopts it. The MacBook runs the same version as the iMac and, thus, after having discovered the AWDL cluster, it also adopts the iPhone as the master node. The iPad briefly adopts the existing master, but then immediately takes over this role as it selects a higher self metric than the iPhone: Fig. 10 shows the current self metric of each node over time. We show the initial value of 60 and the implemented ranges for the different versions of AWDL. Finally, all nodes successively leave the cluster (Wi-Fi turned off) until only the MacBook remains. Since the

[6]https://pcengines.ch/apu.htm

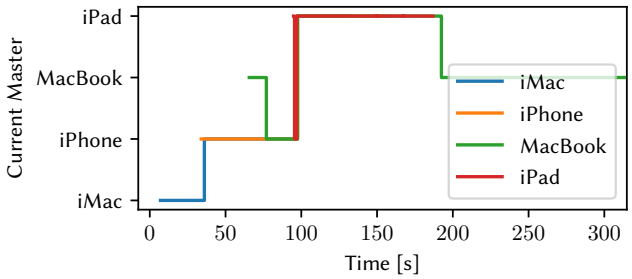

Figure 9: Master selection over time.

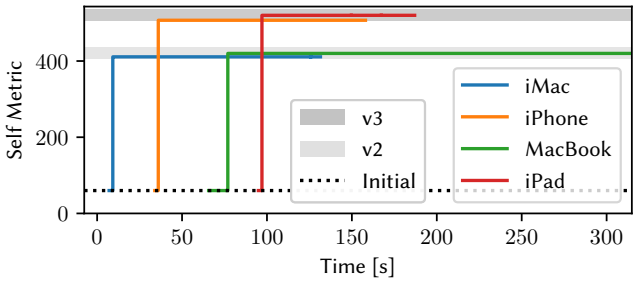

Figure 10: Self metric over time. The grey shaded areas show the value ranges used in the different versions of AWDL.

iMac and the MacBook run an older version of AWDL, they are only selected as master if none of the newer versions are present in the cluster.

Most of these results were expected. What is interesting, however, is that an already existing master node can be "overtaken" by another node running the same version of AWDL. This indicates that Apple's AWDL implementation is rather simplistic: each node keeps the initial self metric only for a short period of time and then selects a higher random value from the version-dependant range *irrespective of whether it has found an existing master or not.*

7.3 Synchronization-to-Master Accuracy

We want to evaluate how well AWDL's master election and synchronization mechanism work. To this end, we monitor the PSF and MIF exchanges between a number of different nodes. We run another idle experiment over a longer period of time (20 min) with three nodes. Figure 11 shows the AW sequence number each node advertises. While Fig. 11 indicates that synchronization works in principle (all nodes follow the same AW sequence number incline), we can see that the AW sequence number steps are not perfectly aligned. We are interested in the magnitude of this synchronization offset. We adapt Eq. (1) to compute the synchronization error ξ between a slave $\mathcal{S}$ and its master $\mathcal{M}$. Assuming a constant airtime t_{air} and given two AFs from $\mathcal{S}$ and $\mathcal{M}$ with a

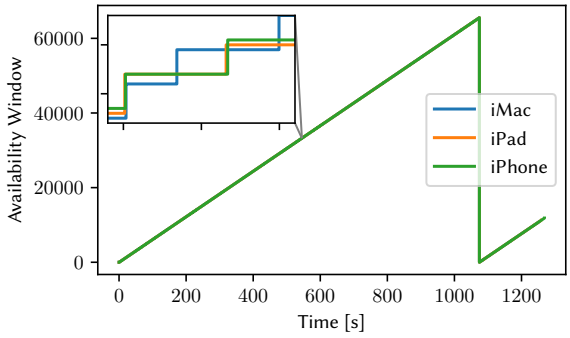

Figure 11: AW sequence number. Showing the sequence number wrap after approximately 18 min ($\approx 2^{16}\,AW \cdot 16\frac{TU}{AW} \cdot 1024\frac{\mu s}{TU}$).

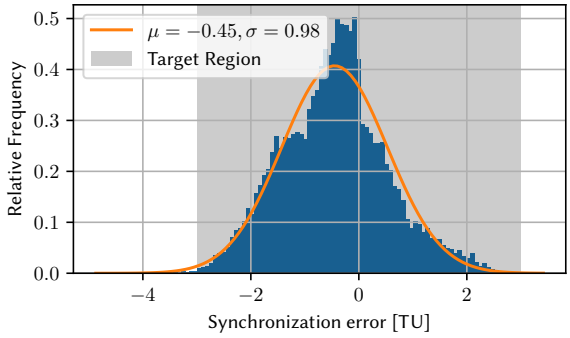

Figure 12: Distribution of synchronization error ξ.

sequence number in the same EAW recorded at the sniffer at time $T^{\mathcal{M}}_{\text{Rx}}$ and $T^{\mathcal{S}}_{\text{Rx}}$, respectively, we calculate ξ as

$$\begin{aligned}\xi &= T^{\mathcal{M}}_{\text{AW}} - T^{\mathcal{S}}_{\text{AW}} \\ &= \left(t^{\mathcal{M}}_{\text{AW}} - t^{\mathcal{S}}_{\text{AW}}\right) \cdot 1024 - \left(t^{\mathcal{M}}_{\text{Tx}} - t^{\mathcal{S}}_{\text{Tx}}\right) + T^{\mathcal{M}}_{\text{Rx}} - T^{\mathcal{S}}_{\text{Rx}}, \\ &\forall i_{\mathcal{S}}, i_{\mathcal{M}} \text{ with } \lfloor \frac{i_{\mathcal{S}}}{p} \rfloor = \lfloor \frac{i_{\mathcal{M}}}{p} \rfloor.\end{aligned} \tag{3}$$

In Fig. 12, we can see that the synchronization error approximates a Gaussian distribution with a mean value of -0.45, and a standard deviation of 0.98. Figure 12 also shows that the target maximum synchronization error of 3 TUs is met in more than 99 % of all cases.

While the results are within the target region, the relatively large synchronization error leads to the conclusion that only a portion of each EAW can reliably be used for communication and the 3 TUs have to used as a guard interval. In numbers, this means that only $1 - \frac{2 \cdot 3\,TU}{64\,TU} \approx 90.6\,\%$ of the interval can be used for communication. The main source of synchronization error lies in the calculation of the transmission delay t_{Tx}. Equation (1) assumes that $T_{\text{Tx,PHY}}$ is set exactly at the moment when the frame is being transmitted via the

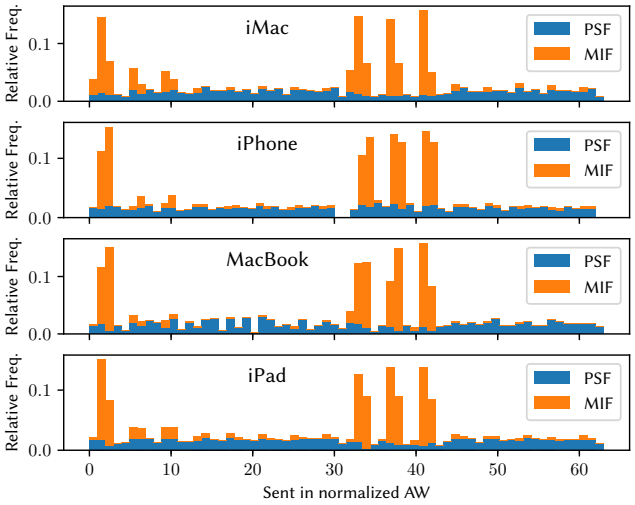

Figure 13: Activity in a full channel sequence period.

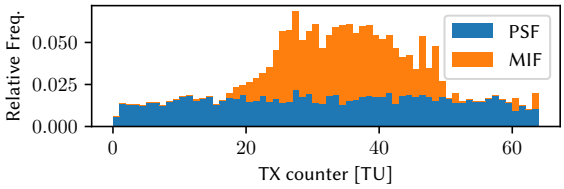

Figure 14: Activity within a single EAW.

Wi-Fi radio after the frame has already been enqueued and additional DCF back-offs have expired. However, we have found that in macOS, $T_{\text{Tx,PHY}}$ is set in the driver right after the AF is created and before DCF has been run. We did not analyze the implementation for other OSes but assume that this is done at a similar location.

7.4 Channel Activity

We want to find out when AFs are usually transmitted. For this, we consider the *idle* scenario from Section 7.2 again. Figure 13 shows when frames (MIF and PSF) are transmitted during an EAW by the different nodes. Each bin represents a single AW (16 TU). We notice that MIFs are mostly sent at the beginning of the first and second half of the entire sequence. We also notice that there is a distinct difference in the sending behavior of MIFs and PSFs. While MIF transmissions adhere to the advertised channel sequence, PSFs are sent at any time. This is probably due to the AF period in the Synchronization Parameters TLV (see Fig. 6) which is either set to 110 or 440 TU and does not align with the 64 AWs that cover one channel sequence. We do not have a solid explanation for this design decision but suspect that it could accelerate the bootstrapping of new nodes which have

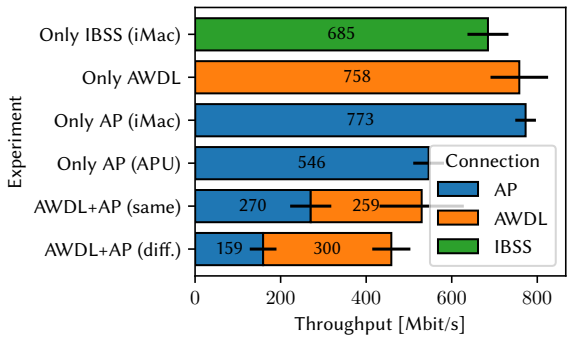

Figure 15: Throughput measurements.

not yet synchronized to a master node. As a downside, this means that nodes cannot really go to a power-conserving mode in a non-advertised slot, which we assumed to be one of the core design goals of AWDL. Another interesting aspect is that PSFs are sent by *all* nodes, no matter if they are master or not. This is another indicator that energy efficiency was not a primary goal of AWDL. Otherwise, only the master and sub-masters would send PSFs.

Figure 13 also shows that the PSFs constitute a certain baseline "noise," while the MIFs are sent especially during the middle of one EAW. Figure 14 "zooms in" and depicts the channel activity within a single EAWs. We see that MIF activity is clustered around the center, while PSFs are sent with equal probability over the entire EAW. We think that MIFs are considered more important since they contain more information than PSFs (see Table 1) and sending in the middle of an EAW increases the chance that a node receives a transmission even if they are not perfectly synchronized.

7.5 Throughput and Channel Hopping

We want to evaluate the impact of AWDL's channel hopping on the throughput of a TCP connection. Unfortunately, Apple drops packets for regular TCP and UDP servers that directly bind to the `awdl0` interface. This meant that running measurement software such as `iperf` was not immediately possible. As a solution, we built an AWDL–TCP proxy via the `NSNetService` API [6] which whitelists the advertised port. In essence, the proxy server advertises a service via DNS-SD and listens for incoming TCP connections. The proxy client component connects to it. Both proxy endpoints also allow TCP connections via the loopback interface such that regular TCP services can simply connect to the loopback interface, and forward the TCP traffic via the `NSNetService` connection. The proxy tool is available at [32].

TCP Throughput. We measure the throughput with `iperf` using three different nodes (MacBook, iMac, and an AP) in six different settings: (1) a single connection from MacBook to the AP *without* AWDL; (2) a single connection from MacBook to iMac via AWDL *without* the AP; (3) two concurrent connections as a combination of (1) and (2), while the AP operates on channel 44; and (4) as (3) but the AP operates on channel 36. Our sniffer is configured as an AP in this scenario which supports a maximum PHY data rate of 866.7 Mbit/s (MCS 9, two streams, 80 MHz bandwidth). iMac and MacBook both support three streams. Thus, we include another measurement (5) where the iMac acts as the AP to see possible throughput differences between an AWDL and an AP connection using the same hardware. Finally, we include (6) a comparison to IEEE 802.11 IBSS mode. We repeat each 10-second experiment 50 times for each setting and show the results in Fig. 15. The error bars indicate the standard deviation. The *only AWDL* and *only AP (iMac)* settings result in similar throughput demonstrating that bandwidth is only limited by the hardware capabilities of the communicating nodes. Note that the *only IBSS (iMac)* setting performs 10–12 % worse than the previous two settings: we observed that the MCS selection mechanism for IBSS on macOS is erratic and does not always choose the maximum supported values even when the signal-to-noise ratio is high. The Qualcomm Wi-Fi chips in the APU only support two streams, so the maximum bandwidth is reduced by approximately 30 %. The cumulative throughput when the AP operates on channel 44 (*same*) is similar to the throughput of the *only AP* setting while the bandwidth between the two connections is uniformly distributed. When the AP operates on a *different* channel, the cumulative throughput drops by about 13 %. This confirms the intuitive assumption that channel switching affects throughput negatively. We are surprised to see that the bandwidth is no longer uniformly distributed between the two streams. Instead, AWDL has a higher throughput which could be caused by AWDL resorting to using all three available streams.

Channel Hopping. We found that AWDL adopts its channel sequence according to the traffic volume on the interface. When there is no traffic (such as in the *idle* scenario), AWDL allocates at least 25 % of the channel sequence to the social channels (see slots 1, 9, 10, and 11 in Fig. 13). As the load increases, AWDL may allocate all EAWs for itself. We depict the various channel allocation states in Table 2.[7] The table shows that (1) at least 25 % of the time is allocated for AWDL (*low power* state), (2) there is *always* a switch to channel 6 in slot 9 possibly for backward compatibility, and (3) at least 25 % of the time is reserved for the AP connection if the node is connected to an AP. In our throughput experiment, either the *data* or the *data+infra* (50 %) state was active.

[7] We found references for 25 states in total (including a *real-time* mode and different combinations) during binary analysis, which we will not further discuss in this paper.

Table 2: A subset of AWDL states and corresponding channel list where p and s are the primary (44) and secondary (6) AWDL channels, respectively, and i is the channel of the AP.

State	Airtime	Channel List ($c = 16$)
Low Power	25.0 %	p s p p
Idle	37.5 %	p p p s p p
Data+Infra	50.0 %	p p p p i i i i s p p p i i i i
	75.0 %	p p p p p p i i s p p p p p i i
Data	100.0 %	p p p p p p p p s p p p p p p p

8 DISCUSSION

In this section, we discuss AWDL complexity and overhead, energy efficiency, and conduct an initial security assessment of AWDL and its OS integration.

8.1 Complexity and Overhead

AWDL has a complex protocol definition that supports various configurations using AWs and EWs. We were surprised to see that Apple settled for a static and rather simple configuration, making the complex concepts obsolete. In addition, we found a lot of redundant information that bloats the size of the AWDL AFs.

(Extended) Availability Windows. AWDL, as implemented in current OSes, allows for highly configurable operation configurations (see Synchronization Parameters TLV in Fig. 6). However, all current implementations use a fixed channel sequence length of 16 and do not differentiate between AWs and EWs but exclusively use the longer EAWs (compare Fig. 7). The reason why Apple prefers EAWs might have to do with the time that is required to perform a channel switch operation in the Wi-Fi chip. We found that a channel switch operation takes at least 8 ms (≈ 8 TU) using `dump chanswitch` of the `wl` utility. In combination with a guard interval that is necessary to cope with the accepted synchronization error of 3 TU, this would leave only 2 TU airtime for communication assuming that the EWs are reserved for an energy conserving *sleep* state. When using EAWs, the temporal efficiency increases from about 12.5 % to more than 78 % while sacrificing opportunities to save energy. We visualize this difference in Fig. 16.

Redundancy. AWDL AFs contain redundant information such as the current master address which is announced in the Synchronization Parameters, Election Parameters, and Election Parameters v2 TLVs. The Service Response Parameters TLV often encode the same information multiple times such that the service instance string and device name be seen three times in a frame when AirDrop is active.

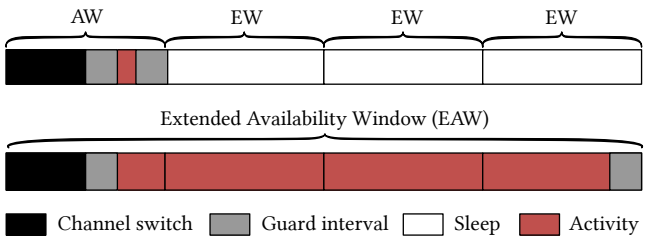

Figure 16: Time spent for channel switching, guard interval, and resulting airtime that can be used for communication when using AWs/EWs vs. EAWs.

8.2 Energy Efficiency

Our working hypothesis was that energy efficiency was one of the primary design goals of AWDL (compare Section 4). The insights obtained from our experimental analysis do not support this hypothesis. We have found that even in the so-called *low power* state, AWDL is active for at least 25 % of the time during which the Wi-Fi chip is active. In addition, all nodes and not only the master send PSFs. We suspect that energy efficiency was sacrificed for a more reliable operation: the exclusive use of long EAWs makes the system more robust against synchronization error. As all nodes send PSFs, new nodes can discover an existing AWDL cluster faster.

8.3 Security

AWDL connections are completely unsecured. However, Apple employs a default packet filter that prevents services to listen on the AWDL interface accidentally. We further found and reported a vulnerability in the macOS driver interface.

Open AWDL Connection. We have found that AWDL connections do not feature any security mechanism. All action and data frames are sent in plain and without authentication. AWDL delegates security functions to the transport and application layer, e. g., AirDrop uses TLS 1.2 [5]. The approach appears to be an informed decision to implement application-dependant policies: a device might be trusted for sending an image file via AirDrop, but not for remote-controlling a Keynote presentation.

Default Packet Filter. While an AWDL connection can be considered insecure, Apple made sure that other services such as file sharing are *not* advertised via the `awdl0` interface which would otherwise be accessible by unauthenticated nearby adversaries. Developers need to explicitly use a dedicated API (e. g., `NSNetService`) to opt-in for the use of AWDL which we did to implement our TCP proxy. The packet filter is apparently not part of the standard macOS firewall but probably implemented in `NSNetService`. Also, the `awdl0` interface is activated only on demand and deactivated once no more traffic is registered, thus, minimizing

the time window for an attack. This could be considered an "accidental" security mechanism because the main reason for the timeout was probably energy conservation.

Vulnerable Driver Interface. The `ioctl` interface described in Section 3.1, especially including the card-specific command used for the Broadcom `wl` utility, could be used by *any local user* on macOS. The issue was reported to Apple on July 19, 2017, and was assigned CVE-2017-13886. Apple has fixed this issue on December 6, 2017, and published the CVE entry on May 2, 2018 [3].

9 CONCLUSION

We reconstructed the frame format and the operation of AWDL, a complex undocumented protocol and complemented our findings with an open source Wireshark dissector. We believe that public knowledge of such wide-spread proprietary protocols is vital to assist wireless network operators and to allow independent security audits as well as to stimulate innovation and research below the application layer. We experimentally evaluated AWDL and showed that the synchronization accuracy is about -0.45 ms on average. The maximum achievable throughput is only limited by the devices' supported PHY data rates if the nodes are not actively using an infrastructure network. If channel switching is required, the cumulative throughput of two concurrent connections drops by about 13 %. We have found a security bug which allowed any local user to access the macOS Wi-Fi driver interface. In the light of recent over-the-air exploitable IEEE 802.11 implementations [9], we suspect that there are even more vulnerabilities to be found given the complexity of the AWDL protocol. As future work, we will direct our efforts towards an energy model for AWDL to understand the implications when using AWDL as a drop-in replacement for BLE or IEEE 802.11 IBSS in ad hoc communication applications.

ACKNOWLEDGMENTS

This work is funded by the LOEWE initiative (Hesse, Germany) within the NICER project and by the German Federal Ministry of Education and Research (BMBF) and the State of Hesse within CRISP-DA.

REFERENCES

[1] Apple Inc. 2015. System Integrity Protection Guide. Retrieved June 28, 2018 from https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/Introduction/Introduction.html

[2] Apple Inc. 2018. About Entitlements. Retrieved June 28, 2018 from https://developer.apple.com/library/archive/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/AboutEntitlements.html

[3] Apple Inc. 2018. About the Security Content of macOS High Sierra 10.13.2, Security Update 2017-002 Sierra, and Security Update 2017-005 El Capitan. Retrieved June 28, 2018 from https://support.apple.com/en-us/HT208331

[4] Apple Inc. 2018. Financial Information–Earnings Releases and 10-K Annual Reports. Retrieved June 28, 2018 from http://investor.apple.com/financials.cfm

[5] Apple Inc. 2018. iOS Security Guide—White Paper. Retrieved June 28, 2018 from https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[6] Apple Inc. 2018. NSNetService Class Documentation. Retrieved June 28, 2018 from https://developer.apple.com/documentation/foundation/nsnetservice

[7] Nitay Artenstein. 2017. Broadpwn: Remotely Compromising Android and iOS via a Bug in Broadcom's Wi-Fi Chipsets. Retrieved June 28, 2018 from https://blog.exodusintel.com/2017/07/26/broadpwn/

[8] Gal Beniamini. 2017. Over the Air: Exploiting Broadcom's Wi-Fi Stack (Part 1). Retrieved June 28, 2018 from https://googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html

[9] Gal Beniamini. 2017. Over The Air: Exploiting Broadcom's Wi-Fi Stack (Part 2). Retrieved June 28, 2018 from https://googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_11.html

[10] Bluetooth Special Interest Group. 2016. Bluetooth® Core Specification.

[11] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. 2013. Device-to-Device Communications with Wi-Fi Direct: Overview and Experimentation. *IEEE Wireless Communications* 20, 3 (2013), 96–104. https://doi.org/10.1109/MWC.2013.6549288

[12] Daniel Camps-Mur, Eduard Garcia Villegas, Elena López-Aguilera, Paulo Loureiro, Paul Lambert, and Ali Raissinia. 2015. Enabling Always On Service Discovery: WiFi Neighbor Awareness Networking. *IEEE Wireless Communications* 22, 2 (2015), 118–125. https://doi.org/10.1109/MWC.2015.7096294

[13] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. 2018. Anatomy of a Vulnerable Fitness Tracking System: Dissecting the Fitbit Cloud, App, and Firmware. *Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 1 (March 2018), 5:1–5:24. https://doi.org/10.1145/3191737

[14] Craig Dooley and Duy Phan. 2017. What's New in Core Bluetooth. In *Worldwide Developers Conference (WWDC)*. Apple Inc. https://developer.apple.com/videos/play/wwdc2017/712/

[15] GitHub. 2015. Leaked BCM4360 Driver Code. Retrieved June 28, 2018 from https://github.com/kyuhsim/khsim_repository/tree/72708c6709/FutureSys/FutureProj_20141222/hg_clone/D700_wl

[16] GitHub. 2018. Lilu: Arbitrary kext and process patching on macOS. Retrieved June 28, 2018 from https://github.com/acidanthera/Lilu

[17] Google. 2017. Wi-Fi Aware. Retrieved June 28, 2018 from https://developer.android.com/guide/topics/connectivity/wifi-aware

[18] Arnt Gulbrandsen, Paul Vixie, and Levon Esibov. 2000. A DNS RR for Specifying the Location of Services (DNS SRV). *RFC* 2782 (Feb. 2000). https://doi.org/10.17487/RFC2782

[19] Robert M. Hinden and Stephen E. Deering. 2006. IP Version 6 Addressing Architecture. *RFC* 4291 (Feb. 2006). https://doi.org/10.17487/RFC4291

[20] IEEE. 2018. Registration Authority. Retrieved June 28, 2018 from https://standards.ieee.org/develop/regauth/index.html

[21] Florian Kohnhäuser, Milan Stute, Lars Baumgärtner, Lars Almon, Stefan Katzenbeisser, Matthias Hollick, and Bernd Freisleben. 2017. SEDCOS: A Secure Device-to-Device Communication System for Disaster Scenarios. In *IEEE Conference on Local Computer Networks (LCN)*.

[22] David Kreitschmann. 2018. User Manual for the Apple CoreCapture Framework. https://seemoo.de/corecapture-manual

[23] David Kreitschmann and Milan Stute. 2018. AWDL and CoreCapture Wireshark dissector. https://seemoo.de/wireshark-awdl

[24] Joakim Linde, Aarti Kumar, Christiaan A. Hartman, and Pierre B. Vandwalle. 2016. WiFi Real-time Streaming and Bluetooth Coexistence. *U.S. Patent* 9485778 (Nov. 2016). https://patents.google.com/patent/US9485778

[25] Zongqing Lu, Guohong Cao, and Thomas La Porta. 2016. Networking Smartphones for Disaster Recovery. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

[26] Ouanilo Medegan. 2012. Skype Reverse Engineering. Retrieved June 28, 2018 from http://www.oklabs.net/skype-reverse-engineering-the-long-journey/

[27] Microsoft. 2018. About the Wireless Ad Hoc API. https://msdn.microsoft.com/en-us/library/windows/desktop/ms705973%28v=vs.85%29.aspx

[28] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2018. The Nexmon Firmware Analysis and Modification Framework: Empowering Researchers to Enhance Wi-Fi Devices. *Computer Communications* (2018). https://doi.org/10.1016/j.comcom.2018.05.015

[29] Serval Project. 2014. Serval Mesh Supported Devices. http://developer.servalproject.org/dokuwiki/doku.php?id=content:servalmesh:supported_devices

[30] IEEE Computer Society. 2014. Standard for Local and Metropolitan Area Networks: Overview and Architecture. https://doi.org/10.1109/IEEESTD.2014.6847097

[31] IEEE Computer Society. 2016. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. https://doi.org/10.1109/IEEESTD.2016.7786995

[32] Milan Stute. 2018. proxAWDL: simple AWDL–TCP proxy. https://seemoo.de/proxawdl

[33] Pierre B. Vandwalle, Tashbeeb Haque, Andreas Wolf, and Saravanan Balasubramaniyan. 2016. Method and Apparatus for Cooperative Channel Switching. *U.S. Patent* 9491593 (Nov. 2016). http://www.google.com/patents/US9491593

[34] Pierre B. Vandwalle, Christiaan A. Hartman, Robert Stacey, Peter N. Heerboth, and Tito Thomas. 2016. Synchronization of Devices in a Peer-to-Peer Network Environment. *U.S. Patent* 9473574 (Oct. 2016). http://www.google.com/patents/US9473574

[35] Wi-Fi Alliance. 2015. Neighbor Awareness Networking Technical Specification.

[36] Wi-Fi Alliance. 2016. Wi-Fi Peer-to-Peer (P2P) Technical Specification. http://www.wi-fi.org/file/wi-fi-peer-to-peer-p2p-technical-specification-v17